Noname manuscript No.
(will be inserted by the editor)

Adversarial frontier stitching for remote neural network watermarking

Erwan Le Merrer · Patrick Pérez · Gilles Trédan

Abstract The state of the art performance of deep learning models comes at a high cost for companies and institutions, due to the tedious data collection and the heavy processing requirements. Recently, [35, 22] proposed to watermark convolutional neural networks for image classification, by embedding information into their weights. While this is a clear progress towards model protection, this technique solely allows for extracting the watermark from a network that one *accesses locally* and entirely.

Instead, we aim at allowing the extraction of the watermark from a neural network (or any other machine learning model) that is operated *remotely*, and available through a service API. To this end, we propose to mark the model's action itself, tweaking slightly its decision frontiers so that a set of specific queries convey the desired information. In the present paper, we formally introduce the problem and propose a novel zero-bit watermarking algorithm that makes use of *adversarial model examples*. While limiting the loss of performance of the protected model, this algorithm allows subsequent extraction of the watermark using only few queries. We experimented the approach on three neural networks designed for image classification, in the context of MNIST digit recognition task.

Keywords Watermarking · Neural network models · Black box interaction · Adversarial examples · Model decision frontiers.

Contact author: Erwan Le Merrer
IRISA/Inria, Campus de Beaulieu, 35 576 Cesson Sévigné, France
Tel.: +33299847213
E-mail: erwan.le-merrer@inria.fr

Patrick Pérez
valeo.ai, Creteil, Île-de-France, France
E-mail: patrick.perez@valeo.com

Gilles Trédan
LAAS/CNRS, 7 avenue du Colonel Roche, 31031 Toulouse, France
E-mail: gtredan@laas.fr

1 Introduction

Recent years have witnessed a fierce competition for the design and training of top notch deep neural networks. The industrial advantage from the possession of a state of the art model is now widely acknowledged, starting to motivate some attacks for stealing those models [33, 9]. Since it is now clear that machine learning models will play a central role in the IT development in the years to come, the necessity for protecting those models appears more salient.

In 1994, [36] proposed to covertly embed a marker into digital content (such as audio or video data) in order to identify its ownership: by revealing the presence of such marker a copyright owner could prove its rights over the content. The authors coined the term digital watermarking. The fact that neural networks are digital content naturally questions the transferability of such techniques to those models.

[35, 22] published the first method for watermarking a neural network that might be publicly shared and thus for which traceability through ownership extraction is important. The marked object is here a neural network and its trained parameters. However, this method requires the ability to directly access the model weights: the model is considered as a white box. The watermark embedding is performed through the use of a regularizer at training time. This regularization introduces the desired statistical bias into the parameters, which will serve as the watermark. We are interested in a related though different problem, namely *zero-bit watermarking* of neural networks (or any machine learning models) that are only remotely accessible through an API. The extraction of a zero-bit watermark in a given model refers to detecting the presence or the absence of the mark in that model. This type of watermark, along with the required *key* to extract it, is sufficient for an entity that suspects a non legitimate usage of the marked model to confirm it or not.

In stark contrast to [35, 22]'s approach, we seek a black box watermarking approach that allows extraction to be con-

ducted remotely, without access to the model itself. More precisely, the extraction test of the proposed watermark consists in a set of requests to the machine learning service, available through an API [33]. This allows the detection of (leaked) models when model's parameters are directly accessible, but also when the model is only exposed through an online service. Second, we target the watermarking of models in general, *i.e.*, our scheme is not restricted solely to neural networks, whether of a certain type or not.

Rationale. We thus aim at embedding zero-bit watermarks into models, that can be extracted remotely. In this setup, we can only rely on interactions with the model through the remote API, *e.g.*, on object recognition queries in case of an image classification model. The input, *e.g.*, images, must thus convey a means to embed identification information into the model (zero-bit watermarking step) and to extract, or not, the identification information from the remote model (watermark extraction step), see Fig. 1. Our algorithm's rationale is that the embedded watermark is a slight modification of the original model's decision frontiers around a set of specific inputs that form the hidden *key*. Answers of the remote model to these inputs are compared to those of the marked model. A strong match (despite the possible manipulation of the leaked model) must indicate the presence of the watermark in the remote model with a high probability.

The inputs in the key must be crafted in a way that watermarking the model of interest does not degrade significantly its performance. To this end, we leverage adversarial perturbations of training examples [11] that produce new examples (the "adversaries") very close the model's decision frontiers. As such adversaries tend to generalize across models, notably across different neural network architectures for visual recognition, see *e.g.*, [29], this frontier tweaking should resist model manipulation and yield only few false positives (wrong identification of non marked models).

Contributions. The contributions of this article are: *1)* A formalization of the problem of zero-bit watermarking a model for remote identification, and associated requirements (Section 2); *2)* A practical algorithm, the *frontier stitching algorithm* based on adversaries that "clamp" the model frontiers, to address this problem. We also introduce a statistical framework for reasoning about the uncertainty regarding the remote model; we leverage a *null hypothesis*, for measuring the success of the watermark extraction (Section 3); *3)* Experiments with three different types of neural networks on the MNIST dataset, validating the approach with regards to the specified requirements (Section 4).

2 Watermarking for Remote Extraction

Considered scenario. The scenario that motivates our work is as follows: An entity, having designed and trained a machine learning model, notably a neural network, wants to zero-bit watermark it (top-action on Fig. 1). This model could then be placed in production for applications and services. In case of the suspicion of a security breach in that application (model has leaked by being copied at a bit-level), the entity suspecting a given online service to re-use that leaked model can query that remote service for answering its doubts (bottom-action).

Like for classic media watermarking methods ([14,36]), our approach includes operations of *embedding* (the entity inserts the zero-bit watermark in its model), and *extraction* (the entity verifies the presence or not of its watermark in the suspected model), and a study of possible *attacks* (actions performed by others in order to remove the watermark from the model).

Modeling Requirements. Following works in the multimedia domain [14], and by [35,22], we adapt the requirements for a watermarking method to the specific capability of *remote* watermark extraction (black box set-up). We choose those requirements to structure the remaining of this article.

We consider the problem of zero-bit watermarking a generic classifier, for remote watermark extraction. Let d be the dimension of the input space (raw signal space for neural nets or hand-crafted feature space for linear and non-linear SVMs), and C the finite set of target labels. Let $k : \mathbb{R}^d \to C$ be the *perfect* classifier for the problem (*i.e.*, $k(x)$ is always the correct answer). Let $\hat{k} : \mathbb{R}^d \to C$ be the trained classifier to be watermarked, and F be the space of possible such classifiers. Our aim is to find a zero-bit watermarked version of $\hat{k}$ (hereafter denoted $\hat{k}_w$) along with a set $K \subset \mathbb{R}^d$ of specific inputs, named the *key*, and their labels $\{\hat{k}_w(x), x \in K\}$. The purpose is to query with the key a remote model that can be either $\hat{k}_w$ or another unmarked model $k_r \in F$. The key, which is thus composed of "objects" to be classified, is used to embed the watermark into $\hat{k}$.

Here are listed the requirements of an *ideal* watermarked model and key couple, $(\hat{k}_w, K)$:

Loyal. The watermark embedding does not hinder the performance of the original classifier:

$$\forall x \in \mathbb{R}^d, \notin K,\ \hat{k}(x) = \hat{k}_w(x). \quad (1)$$

Efficient. The key is as short as possible, as accessing the watermark requires $|K|$ requests.

Effective. The embedding allows unique identification of $\hat{k}_w$ using K (zero-bit watermarking):

$$\forall k_r \in F,\ k_r \neq \hat{k}_w \Rightarrow \exists x \in K \text{ s.t. } k_r(x) \neq \hat{k}_w(x). \quad (2)$$

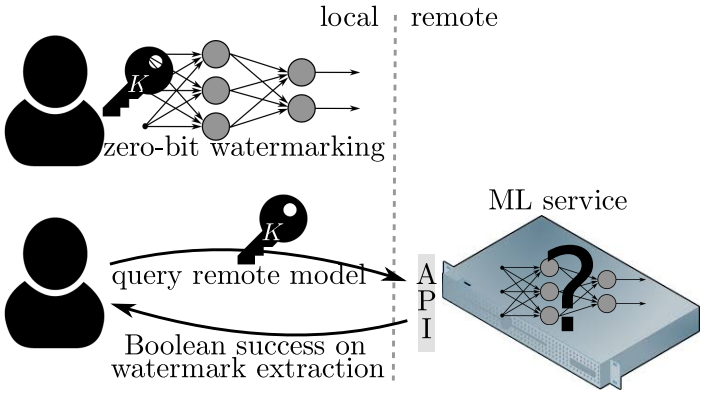


Fig. 1 Our goal: zero-bit watermarking a model locally (top-action), for remote assessment of a potential model leak (bottom-action).

Robust. Attacks (such as fine-tuning or compression) to $\hat{k}_w$ do not remove the watermark[1]:

$$\forall x \in K,\ (\hat{k}_w + \varepsilon)(x) = \hat{k}_w(x). \tag{3}$$

Secure. No efficient algorithm exists to detect the presence of the watermark in a model by an unauthorized party.

Note that *effectiveness* is a new requirement as compared to the list of Uchida *et al*. Also, their *capacity* requirement, *i.e.*, the amount of information that can be embedded by a method, is not part of ours as our goal is to decide whether watermarked model is used or not (zero-bit watermark extraction).

One can observe the conflicting nature of effectiveness and robustness: If, for instance, $(\hat{k}_w + \varepsilon) \in F$ then this function violates one of the two. In order to allow for a practical setup for the problem, we rely on a measure $m_K(a, b)$ of the matching between two classifiers $a, b \in F$:

$$m_K(a, b) = \sum_{x \in K} \delta(a(x), b(x)), \tag{4}$$

where δ is the Kronecker delta. One can observe that $m_K(a, b)$ is simply the Hamming distance between the vectors $a(K)$ and $b(K)$, thus based on elements in K. With this focus on distance, our two requirements can now be recast in a non-conflicting way:

- Robustness: $\forall \varepsilon \approx 0, m_K(\hat{k}_w, \hat{k}_w + \varepsilon) \approx 0$
- Effectiveness: $\forall k_r \in F, m_K(\hat{k}_w, k_r) \approx |K|$

3 The Frontier Stitching Algorithm

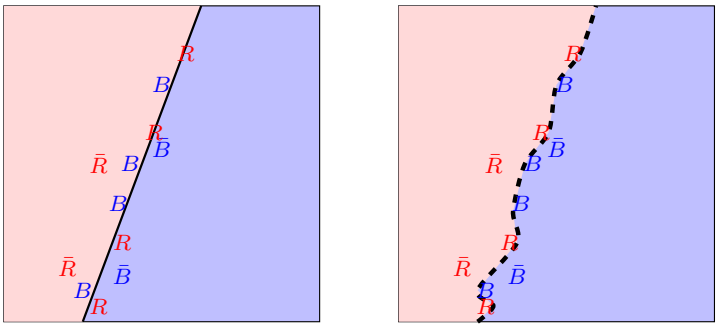
We now present a practical zero-bit model watermarking algorithm that permits remote extraction through requests to an API, following the previously introduced requirements. Our aim is to output a watermarked model $\hat{k}_w$, which can for instance be placed into production for use by consumers, together with a watermark key K to be used in case of model leak suspicion. For the security requirement to hold, we obviously discard any form of *visible* watermark insertion [4]. Fig. 2 illustrates the approach in the setting of a binary classifier (without loss of generality).

As we use input points for watermarking the owned model and subsequently to query a suspected remote model, the choice of those inputs is crucial. A non watermarking-based solution based simply on choosing arbitrarily $|K|$ training examples (along with their correct labels), is very unlikely to succeed in the identification of a specific valuable model: Classifying those points correctly should be easy for highly accurate classifiers, which will then provide similar results, ruining the effectiveness. On the other hand, the opposite strategy of selecting $|K|$ arbitrary examples and fine-tuning $\hat{k}$ so that it changes the way they are classified (*e.g.*, $\forall x \in K,\ \hat{k}(x) \neq \hat{k}_w(x)$) is an option to modify the model's behavior in an identifiable way. However, fine-tuning on even few examples that are possibly far from decision frontiers will significantly alter the performance of $\hat{k}$: The produced solution will not be loyal.

Together, those observations lead to the conclusion that the selected points should be close to the original model's decision frontier, that is, their classification is not trivial and depends heavily on the model (Fig. 2(a)). Finding and manipulating such inputs is the purpose of adversarial perturbations [11,21]. Given a trained model, any well classified example can be modified in a very slight way such that it is now misclassified. Such modified samples are called "adversarial examples", or adversaries in short.

The proposed frontier stitching algorithm, presented in Algorithm 1, makes use of such adversaries, selected to "clamp" the frontier in a unique, yet harmless way (Fig. 2(a)). It proceeds in two steps to mark the model. The first step is to select a small key set K of specific input points, which is composed of two types of adversaries. It first contains classic adversaries, we call *true adversaries*, that are misclassified by $\hat{k}$ although being each very close to a well classified

[1] "$\hat{k}_w + \varepsilon$" stands for a small modification of the parameters of $\hat{k}_w$ that preserves the value of the model, *i.e.*, that does not deteriorate significantly its performance.



(a) (b)

Fig. 2 Illustration of proposed approach for a binary classifier. (a) The proposed algorithm first computes "true adversaries" (R and B) and "false" ones ($\bar{R}$ and $\bar{B}$) for both classes from training examples. They all lie close the decision frontier. (b) It then fine-tunes the classifier such that these inputs are now all well classified, *i.e.*, the 8 true adversaries are now correctly classified in this example while the 4 false ones remain so. This results in a loyal watermarked model (very similar to original one) and a key size of $|K| = 12$ here. This process resembles "stitching" around data points.

example. It also contains *false adversaries*, each obtained by applying an adversarial perturbation to a well classified example without ruining its classification. In practice, the "fast gradient sign method" proposed in [11] is used with a suitable gradient step to create potential adversaries of both types from training examples, these adversaries are inputs that will be closer to a decision frontier than their base inputs. This modification in the direction of other classes is the purpose of adversarial attacks. (Other methods include for instance the "Jacobian-based saliency map approach" [27]).

These frontier clamping inputs are then used to mark the model (Fig. 2(b)). The model $\hat{k}$ is fine-tuned into $\hat{k}_w$ such that all points in K are now well classified:

$$\forall x \in K,\ \hat{k}_w(x) = k(x). \tag{5}$$

In other words, the true adversaries of $\hat{k}$ in K become false adversaries of the marked model, and false adversaries remain as such. The role of the false adversaries is to limit strongly the amount of changes that the decision frontiers will undergo when getting true adversaries back to the right classes. False adversaries also have the role of characterizing the shapes of the model frontiers, for adding robustness to the statistical watermark extraction process we now present.

Statistical watermark extraction. The marking step is thus the embedding of such a crafted key in the original model, while the watermark extraction consists in asking the remote model to classify the inputs in key K, to assess the presence or not of the zero-bit watermark (Algorithm 2). We now analyze statistically this extraction problem.

As discussed in Section 2, the key quantity at extraction time is the Hamming distance m_K (Eq. 4) between remote model's answers to the key and expected answers. The stitching algorithm produces deterministic results with respect to the imprinting of the key: A marked model perfectly matches the key, *i.e.*, the distance m_k between $\hat{k}_w$ and query results in Algorithm $2(K, K_{labels})$ equals zero. However, as the leaked model may undergo arbitrary attacks (*e.g.*, for watermark removal), transforming $\hat{k}_w$ into a model $\hat{k}'_w$, one should expect some deviation in $\hat{k}'_w$ answers to watermark extraction ($0 \leqslant m_K(\hat{k}_w, \hat{k}'_w) \ll |K|$). On the other hand, other unmarked models might also partly match key labels, and thus have a positive non-maximum distance too. As an extreme example, even a strawman model that answers a label uniformly at random produces $|K|/|C|$ matches in expectation, when classifying over $|C|$ classes. Consequently, two questions are central to the frontier stitching approach: *How large is the deviation one should tolerate from the original watermark in order to state about successful zero-bit watermark?* And, dependently, *how large should the key be, so that the tolerance is increased?*

We propose to rely on a probabilistic approach by estimating the probability of an unmarked model k_r to produce correct answers to requests from inputs in the key. While providing an analysis that would both be precise and cover all model behaviors is unrealistic, we rely on a *null-model* assuming that inputs in the key are so close to the frontier that, at this "resolution", the frontier only delimits two classes (the other classes being too far from the considered key inputs), and that the probability of each of the two classes is $1/2$ each. This is all the more plausible since we leverage adversaries especially designed to cause misclassification.

More formally, let $k_\emptyset$ be the null-model: $\forall x \in K,\ \mathbb{P}[k_\emptyset(x) = \hat{k}_w(x)] = 1/2$. Having such a null-model allows applying a p-value approach to the decision criteria. Indeed, let $Z = m_K(\hat{k}_w, k_r)$ be the random variable representing the number of mismatching labels for key inputs K. Assuming that the remote model is the null-model, the probability of having exactly z errors in the key is $\mathbb{P}[Z = z | k_r = k_\emptyset] = 2^{-|K|}\binom{|K|}{z}$, that is Z follows the binomial distribution $B(|K|, \frac{1}{2})$. Let θ be the maximum number of errors tolerated on k_r's answers to decide whether or not the water-

Algorithm 1 Zero-bit watermarking a model

Require: Labelled sample set (X, Y); Trained model $\hat{k}$; Key length $\ell = |K|$; Step size ε for adversary generation;
Ensure: $\hat{k}_w$ is watermarked with key K {Assumes X is large enough and ε is balanced to generate true & false adversaries}
{Key construction}
$adv\_candidates \leftarrow$ GEN_ADVERSARIES($\hat{k}$,(X,Y),ε)
while $|key_{true}| < \ell/2$ or $|key_{false}| < \ell/2$ **do**
  pick random adversary candidate $c \in adv\_candidates$, associated to $x \in X$ with label y_x
  if $\hat{k}(x) = y_x$ and $\hat{k}(c) \neq y_x$ and $|key_{true}| < \ell/2$ **then** {c is a true adversary}
    $key_{true} \leftarrow key_{true} \cup \{(c, y_x)\}$
  else if $\hat{k}(c) = \hat{k}(x) = y_x$ and $|key_{false}| < \ell/2$ **then** {c is a false adversary}
    $key_{false} \leftarrow key_{false} \cup \{(c, y_x)\}$
  end if
end while
$(K, K_{labels}) \leftarrow key_{true} \cup key_{false}$
{Force embedding of key adversaries in their original class}
$\hat{k}_w \leftarrow$ TRAIN($\hat{k}$, K, K_{labels})
return $\hat{k}_w$, K, K_{labels}

Algorithm 2 Zero-bit watermark extraction from a remote model

Require: K and K_{labels}, the key and labels used to watermark the neural network
$m_K \leftarrow 0$
for each $c \in K$ **do**
  if QUERY_REMOTE(c) $\neq K_{labels}(c)$ **then**
    $m_K \leftarrow m_K + 1$ {remote model answer differs from recorded answer}
  end if
end for
{Having θ such that $2^{-|K|} \sum_{z=0}^{\theta} \binom{|K|}{z} < 0.05$ (*null-model*)}
return $m_K < \theta$ {True $\Leftrightarrow$ successful extraction}

mark extraction is successful. To safely (p-value < 0.05) reject the hypothesis that k_r is a model behaving like our null-model, we need $\mathbb{P}[Z \leq \theta | k_r = k_\emptyset] < 0.05$. That is $2^{-|K|} \sum_{z=0}^{\theta} \binom{|K|}{z} < 0.05$. In particular, for key sizes of $|K| = 100$ and $|K| = 20$ a p-value of 0.05, the maximum number of tolerated errors are $\theta = 42$ and 6, respectively. We thus consider the zero-bit watermark extraction from the remote model successful if the number of errors is below that threshold θ, as presented in Algorithm 2. Next Section includes an experimental study of false positives related to this probabilistic approach.

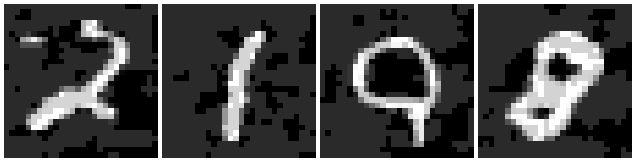
4 Experiments

We now conduct experiments to evaluate the proposed approach in the light of the requirements stated in Section 2. In particular, we evaluate the fidelity, the effectiveness and the robustness of our algorithm.

We perform our experiments on the MNIST dataset [18], using the Keras backend [7] to the TensorFlow platform[2] [1]. As neural network architectures, we use three off-the-shelf implementations, available publicly on the Keras website, namely `mnist_mlp` (0.984% accuracy at 10 epochs, we denote as MLP), `mnist_cnn` (0.993% at 10, denoted as CNN) and `mnist_irnn` (0.9918% at 900, denoted as IRNN). Their characteristics are as follows. The MLP is composed of two fully connected hidden layers of 512 neurons each, for a total of $669,706$ parameters to train. The CNN is composed by two convolutional layers (of size 32 and 64), with kernel sizes of 3×3, followed by a fully connected layer of 128 neurons (for a total of $710,218$ parameters). Finally, the IRNN refer to settings by Le *et al.* [15] and uses a fully connected recurrent layer (for a total of $199,434$ parameters). All three architectures use a softmax layer as output.

All experiments are run on networks trained with the standard parametrization setup: MNIST training set of $60,000$ images, test set of size $10,000$, SGD with mini-batches of size 128 and a learning rate of 0.001.

[2] Code will be open-sourced on GitHub, upon article acceptance.



(a) (b) (c) (d)

Fig. 3 An example of a key K of size four (generated with the "fast gradient sign method" and $\varepsilon = 0.1$), for the CNN classifier and the MNIST task: Key inputs (a) and (b) are true adversaries (2 and 1 digits, classified as 9 and 8 respectively), while (c) and (d) are false adversaries (9 and 8 digits that are correctly classified but knowingly close to a decision frontier).

4.1 Generating adversaries for the watermark key.

We use the Cleverhans Python library by [24] to generate the adversaries (function GEN_ADVERSARIES() in Algorithm 1). It implements the "fast gradient sign method" by [11], we recall here for completeness. With θ the parameters of the attacked model, $J(\theta, x, y)$ the cost function used to train the model and $\bigtriangledown$ the gradient of that cost function with respect to input x, the adversarial image x^* is obtained from the input image x by applying the following perturbation:

$$x^* = x + \varepsilon \, . \, sign(\bigtriangledown_x J(\theta, x, y)).$$

ε thus controls the intensity of the adversarial perturbation; we use a default setting of $\varepsilon = 0.25$.

Alternative methods, such as the "Jacobian-based saliency map" ([26]), or other attacks for generating adversaries may also be used (please refer to [38] for a list of existing approaches). Adversaries are crafted from some test set images that are then removed from the test set to avoid biased results.

As explained in Section 3, our key set K is composed by 50% of true adversaries, and by 50% of false adversaries (adversarial perturbations that are not causing misclassification). In the fast gradient sign method, the value of ε controls the intensity of the perturbations that are applied to the attacked images. With a large value, most of the adversaries are true adversaries and, conversely, a small ε produces mostly false adversaries. As a consequence, ε must be chosen so that Algorithm 1 as enough inputs ($\ell/2$) of each kind, in order to build the key. Altogether, the adversaries in the key are close to the decision frontiers, so that they "clamp" these boundaries. An example of a key is displayed in Fig. 3.

4.2 Impact of watermarking (fidelity requirement).

This experiment considers the impact on fidelity of the watermark embedding, of sizes $|K| = 20$ and $|K| = 100$, in the three networks. We generated multiple keys for this experiment and the following ones (see Algorithm 1), and kept those which required fewer that 100 epochs for embedding in the models (resp. 1000 for IRNN), using a fine tuning rate of $\frac{1}{10}$th of the original training rate. The following results are averaged over 30 independent markings per network.

The cumulative distribution function (CDF) in Fig. 4 shows the accuracy for the 3 networks after embedding keys of the two sizes. IRNN exhibits nearly no degradation, while embedding in the MLP causes on average 0.4% and 0.8% loss for respectively key sizes 20 and 100.

We remarked no significant degradation difference when marking ($|K| = 20$, 10 independent runs) a model with adversaries generated under ℓ_1, ℓ_2 and ℓ_∞ norms. For instance, we marked the CNN model with $\varepsilon = 150$ (ℓ_1, fooling 88.54% of MNIST test set), $\varepsilon = 7$ (ℓ_2, fooling 87.56%) and $\varepsilon = 0.25$ (ℓ_∞, fooling 89.08%); This has resulted in accuracy drops of respectively 0.23%, 0.22% and 0.14%. We use ℓ_∞ in the sequel.

4.3 False positives in remote watermark extraction (effectiveness requirement).

We now experiment the effectiveness of the watermark extraction (Algorithm 2). When querying the remote model returns True, it is important to get a low false positive rate. To measure this, we ran on *non watermarked* and retrained networks of each type the extraction Algorithm 2, with keys used to watermark the three original networks. Ideally, the output should always be negative. We use $|K| = 100$, and various values of $\varepsilon \in \{0.025, 0.1, 0.25, 0.5\}$. We observe on Fig. 6 that the false positives are occurring for lower values 0.025 and 0.1 on some scenarios. False positives disappear for $\varepsilon = 0.5$.

This last experiment indicates that the model owner has to select a high ε value, depending on her dataset, as the generated adversaries are powerful enough to prevent accurate classification by the remote inspected model. We could not assess a significant trend for a higher degradation of the marked model when using a higher ε as depicted in Fig. 5, where the CNN model is marked with keys of $|K| = 20$ and for $\varepsilon \in \{0.025, 0.1, 0.5\}$ (10 runs per ε value). The 0.25 value is to be observed on Fig. 4. We note that this relates to *adversarial training*, a form of specialized data augmentation that can be used as generic regularization [11] or to improve model resilience to adversarial attacks [24]. Models are thus trained with adversarial examples, which is in relation to our watermarking technique that incorporates adversarial examples in a fine-tuning step, without ruining model accuracy.

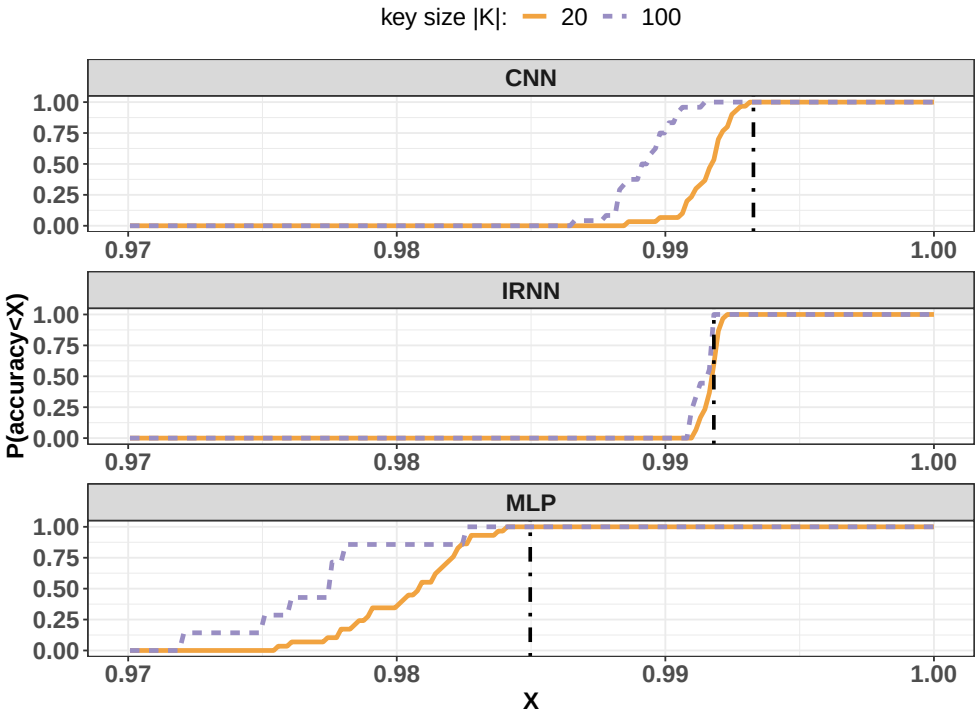


Fig. 4 Distribution of the degradation caused by watermarking models (resulting accuracy), with multiple keys of size 20 and 100 ($\varepsilon = 0.25$). Black vertical dot-dash lines indicate pre-watermarking accuracies.

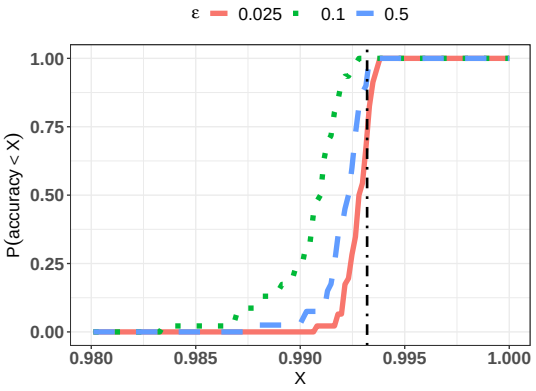


Fig. 5 Distribution of the degradation caused by watermarking the CNN model (resulting accuracy), with multiple keys of size 20, and for various $\varepsilon \in \{0.025, 0.1, 0.5\}$ in the ℓ_∞ norm. The black vertical dot-dash line indicates the pre-watermarking accuracy.

4.4 Attacking the watermarks of a leaked model (robustness requirement).

We now address the robustness of the stitching algorithm. Two types of attacks are presented: Model compression (via both *pruning* and *singular value decomposition*) and overwriting via fine-tuning.

We consider *plausible* attacks over the leaked model, *i.e.*, attacks that do not degrade the model beyond a certain accuracy, which we set to 0.95 in the sequel[3] (the three networks in our experiments have accuracy above 0.984).

In our set-up, re-using a leaked model that has been significantly degraded in the hope to remove a possible watermark does not make sense; the attacker would rather use a less precise, yet legitimate model.

We remark that due to the nature of our watermarking method, an attacker (who does not possess the watermark key) will not know whether or not her attacks removed the watermark from the leaked model.

Compression attack via pruning As done by [35], we study the effect of compression through parameter pruning, where 25% to 85% of model weights with lowest absolute values are set to zero. Results are presented on Tab. 1. Among all plausible attacks, none but one (50% pruning of IRNN parameters) prevents successful and 100% accurate extraction of the watermarks. We note that the MLP is prone to important degradation of accuracy when pruned, while at the same time the average number of erased key elements from the model is way below the decision threshold of 42. Regarding the CNN, even 85% of pruned parameters are not enough to reach that same threshold.

Compression attack via Singular Value Decomposition We experimented with a second compression attack: we used a compression library available on GitHub (`keras`

[3] This about 3.5% accuracy drop is also the one tolerated by a recent work on *trojaning* neural networks [20].

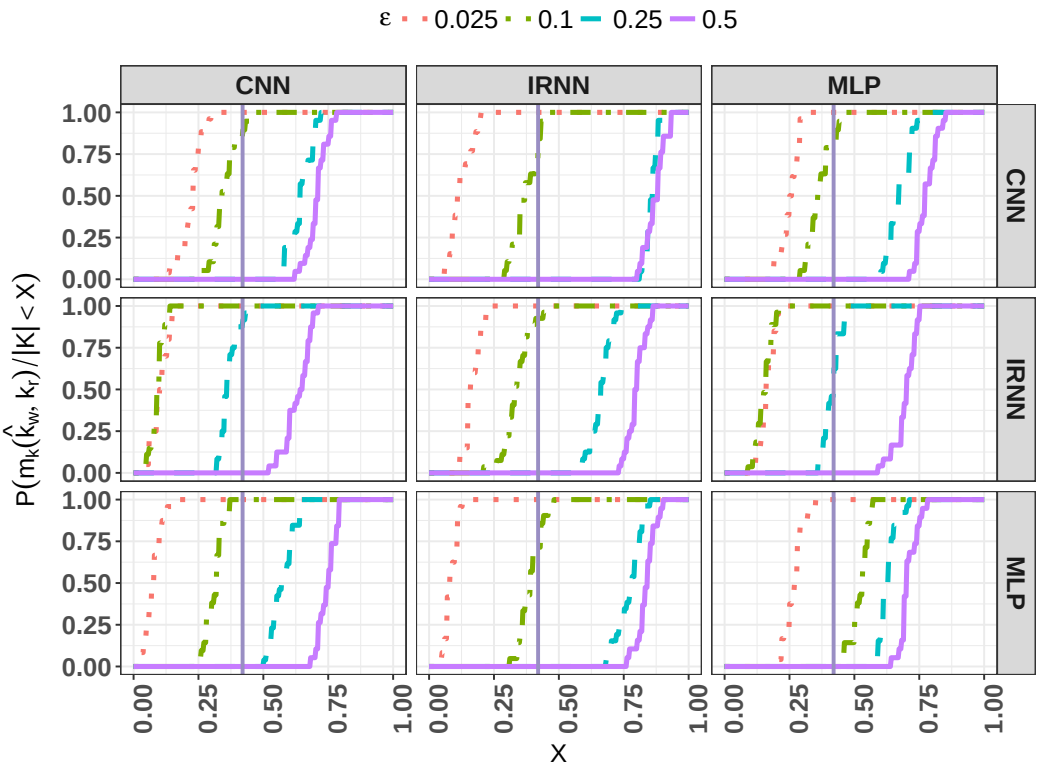


Fig. 6 Distribution of (normalized) Hamming distance when extracting the watermark (generated on networks $\hat{k}_w$ listed on right axis) from remote unmarked models k_r (w. names on the top), for $|K| = 100$ and four values of ε. Vertical solid bars indicate the decision threshold corresponding to $p < .05$ criteria. False positives (*i.e.*, unmarked model at distance less than the threshold) happen when the CDF is on the left of the bar. Algorithm 1's value $\varepsilon = 0.5$ (purple solid curve) prevents the issue of false positives in all cases.

	Pruning rate	K elts removed	Stdev	Extraction rate	Acc. after
CNN	0.25	0.053/100	0.229	1.000	0.983
-	0.50	0.263/100	0.562	1.000	0.984
-	0.75	3.579/100	2.479	1.000	0.983
-	0.85	34.000/100	9.298	0.789	0.936
IRNN	0.25	14.038/100	3.873	1.000	0.991
-	0.50	59.920/100	6.782	0.000	0.987
-	0.75	84.400/100	4.093	0.000	0.148
MLP	0.25	0.360/100	0.700	1.000	0.951
-	0.50	0.704/100	0.724	1.000	0.947
-	0.75	9.615/100	4.392	1.000	0.915
-	0.80	24.438/100	5.501	1.000	0.877

Table 1 Robustness to compression attack (pruning-based): Watermark extraction success rates ($|K| = 100$), after a pruning attack on watermarked models. Different pruning rates are tested to check if watermarks get erased while accuracy remains acceptable. Results in gray rows are to be ignored (not plausible attacks).

`compressor`[4]) which leverages *singular value decomposition* (SVD) on model weights, to compress them. We tested it to be compatible with MLP and CNN. Tab. 2 shows that the key extraction on CNN is not affected by this new attack, as accuracy drops at 88% for 50% weights compression and with an average of 20 elements removed (over 42 tolerated). The extraction from MLP starts to be affected with high compression of 75% of the weights, with 48 elements removed.

Overwriting attack via adversarial fine-tuning Since we leverage adversaries in the key to embed the watermark in the model, a plausible attack is to try overwriting this watermark via adversarial fine-tuning of the leaked model. As explained in Section 4.3, this action also relates to adversarial training that originally aims to improve model resilience to adversarial attacks [24]. In this experiment, we turn $1,000$ images from the MNIST test set into adversaries and use

[4] `https://github.com/DwangoMediaVillage/keras_compressor`

	Pruning rate	K elts removed	Stdev	Extraction rate	Acc. after
CNN	0.25	1.867/100	1.457	1.000	0.987
-	0.50	20.143/100	4.721	1.000	0.885
-	0.75	69.643/100	3.296	0.000	0.426
-	0.90	83.714/100	3.989	0.000	0.255
MLP	0.25	0.385/100	0.898	1.000	0.973
-	0.50	5.760/100	2.697	1.000	0.966
-	0.75	48.423/100	5.742	0.077	0.953
-	0.90	83.760/100	6.346	0.000	0.157

Table 2 Robustness to compression from library `keras_compressor` (SVD-based), compatible with CNN and MLP.

	K elts removed	Stdev	Extraction rate	Acc. after
CNN	17.842	3.594	1.000	0.983
IRNN	37.423	3.931	0.884	0.989
MLP	27.741	5.749	1.000	0.972

Table 3 Robustness to overwriting attacks: Rate of remaining zero-bit watermarks in the three attacked models, after model fine-tuning with $1,000$ new adversaries.

them to fine-tune the model (the test set being the remaining $9,000$ images). The results of the overwriting attacks is presented on Tab. 3. An adversarial fine-tuning of size $1,000$ uses 20 times more adversaries than the watermarking key (as $|K| = 100$, with 50% true adversaries). We see perfect watermark extractions (no false negatives) for CNN and MLP, while there are few extraction failures from the attacked IRNN architecture. This experiment is thus consistent with recent arguments about the general difficulty to defend against adversaries [34, 31].

Conclusion on the attacks of watermarks We highlight that in all but two tries, the watermark is robust to considered attacks. In addition, since the attacker cannot know whether her attack is successful or not (as the key is unknown to her), each new attack trial is bound to degrade the model even further, without removal guarantee. This uncertainty will probably considerably discourage trials of this kind.

4.5 About the efficiency and security requirements.

The efficiency requirement deals with the computational cost of querying a suspected remote service with the $|K|$ queries from the watermarking key. Given typical pricing of current online machine learning services (Amazon's Machine Learning, for instance, charges $\$0.10$ per $1,000$ classification requests as per Jan. 2018), keys in the order of hundreds of objects as in our experiments incur financial costs that are negligible, an indication of negligible computational cost as well. It is to be noted that if the suspected model is running on an embedded device [13], there is no cost in querying that model; if we target such an application, keys can then be of arbitrary length as far as their embedding preserve the accuracy of the model to be watermarked. As for the watermarking step (key embedding), it is as complex as fine-tuning a network using set K. Since the size of K is negligible compared to the original training set size, the computational overhead of embedding a key is considered low.

The frontier stitching algorithm deforms slightly and locally the decision frontiers, based on the labelled samples in key K. To ensure security, this key must be kept secret by the entity that watermarked the model (otherwise, one might devise a simple overwriting procedure that reverts these deformations). Decision frontier deformation through fine-tuning is a complex process (see work by [3]) which seems very difficult to revert in the absence of information on the key (this absence also prevents the use of recent statistical defense techniques [12]). Could a method detect specific local frontier configurations that are due to the embedded watermark? The existence of such an algorithm, related to *steganalysis* in the domain of multimedia, would indeed be a challenge for neural network watermarking at large, but seems unlikely.

Our watermarking method relies on algorithms for finding adversarial examples, in order to get inputs nearby decision frontiers. Last few years have witnessed an arms race between attacks (algorithms for producing adversarial examples) and defenses to make neural networks more robust to those attacks; the problem is still open [31] (please refer to [38] for a survey). We argue that our method will remain functional regardless of this arms race, as the purpose of machine learning for classification is to create decision frontiers for separating target classes; there will always be means to cross those frontiers, and then to create the inputs we require in order to create our watermark keys. Authors in [31] precisely characterize classes of problems for which adversarial examples cannot be avoided, and that depend on properties of the data distribution as well as the space dimensionality of the dataset.

5 Related Work

Watermarking aims at embedding information into "objects" that one can manipulate locally. One can consider the insertion of *visible* watermarks in those objects [4]; we consider in this work and related work the insertion of invisible watermarks. Watermarking multimedia content especially is a

rich and active research field, yet showing a two decades old interest [14]. Neural networks are commonly used to insert watermarks into multimedia content [6].

After extension to surprising domains such as network science [40], the extension to watermarking neural networks as objects themselves is new, following the need to protect the valuable assets of today's state of the art machine learning techniques. Uchida *et al.* [35,22] thus propose the watermarking of neural networks, by embedding information in the learned weights. Authors show in the case of convolutional architectures that this embedding does not significantly change the distribution of parameters in the model. Mandatory to the use of this approach is a local copy of the neural network to inspect, as the extraction of the watermark requires reading the weights of convolution kernels. This approach is motivated by the voluntary sharing of already trained models, in case of *transfer learning*, such as in [32]'s work for instance.

Neural network watermarking techniques that allow verification in a remote black-box context were recently proposed. These works relate to our black box system model (as introduced in the technical report [16]). They indeed leverage the model's outputs to carefully chosen inputs to retrieve the watermark. [39] proposes to train the model to be protected with a set of specifically crafted inputs in order to trigger the assignment of a specific target label by the model on those inputs. In other words, their approach is very similar to trojaning [20]: the triggering of a specific label facing a crafted input constitutes for the authors a proof of ownership. Similarly, [2] explicitly exploits the possibility of neural network trojaning (*i.e.*, , backdooring) for the same purpose. Finally, [28] directly embeds the watermark into the layer weights using specific loss functions. However, contrary to [35] for instance where the weights directly contain the watermark (which is therefore not accessible in a black box context), weights in [28] are modified in order to produce desired activations at runtime given specific inputs. Surprisingly, the authors also show that watermarking mechanisms designed in a black box context can also be used in a white box context. In a more restricted setup, Guo *et al.* propose the adaptation to embedded devices of a black box capable watermark extraction [13].

Since more and more models and algorithms might only be accessed through API operations (as being run as a component of a remote online service), there is a growing body of research which is interested in leveraging the restricted set of operations offered by those APIs to gain knowledge about the remote system internals. [33] demonstrates that it is possible to extract an indistinguishable copy of a remotely executed model from some online machine learning APIs.

Depth of neural models may also be infered using *timing side channel attacks* [10]. [25] have shown attacks on remote models to be feasible, yielding erroneous model outputs. Authors in [30] target attacks such as evading a CAPTCHA test, by the reverse engineering of a remote classifier models. Other attacks focus on the stealing of model hyperparameters, from APIs [37,23]; [23] aims at infering inner hyperparameters (*e.g.*, number of layers, non-linear activation type) of a remote neural network model by analysing its response patterns to certain inputs. Finally, algorithms in [17] propose to detect the tampering with a deployed model, also through simple queries to the API; tested attacks are trojaning, compression, fine-tuning and the watermarking method proposed in this paper. In present work, we propose a watermarking algorithm that is compliant with APIs, since it solely relies on the basic classification query to the remote service.

6 Conclusion and perspectives

This article introduces the frontier stitching algorithm, to extract previously embedded zero-bit watermarks from leaked models that might be used as part of remote online services. We demonstrated this technique on image classifiers; sound [5] and video classifiers [19] were also recently found to be prone to adversarial attacks. We believe that a demonstration of existing watermarking techniques in those domains would be of a great practical interest.

We focused on classification problems, which account for many if not most ML-based services. Extensions to other problems (like regressions or semantic segmentation of images) are a next step for future work, since adversarial examples also affect those domains.

Regarding the model architecture aspect, we have seen that the IRNN model is prone to compression attacks (pruning rate of 50% of parameters). This underlines the specific behavior of architectures facing attacks and marking; in depth characterization is an interesting future work.

We challenged the robustness of our watermarking scheme facing compression and overwriting attacks. Other more advanced types of attacks might be of interest for an attacker that wants to remove the inserted watermark. In particular, another attack may be the transfer learning of the watermarked model to another task. Recent work [8] provides a first empirical evidence that the adversaries that were integrated in the model through learning (defense) might not survive the transfer, leading to a potentially successful watermark removal. A full characterization of the resilience of adversaries facing transfer learning attacks is of great importance for future work.

As another future work, we stress that the watermark information is currently extracted using the binary answers to the query made on each object in the key: Whether or not this object is classified by the remote model as expected in the key label. Leveraging not only those binary answers,

but also the actual classification issued by the remote model (or even the classification scores), may allow one to embed more information with the same watermark size. Another possible improvement may come from the use of the recent concept of universal adversarial perturbations ([21]): they might be leveraged to build efficient and robust watermarking algorithms. Indeed, this method generates adversaries that can fool multiple classifiers at once. Relying on such adversaries in an extension of our framework might give rise to new, improved watermarking algorithms for neural networks that are queried in a black box setup.

Finally, we recall that the watermarking technique we proposed, as well as the ones from the related work [39, 2, 13, 28], make the assumption that the watermarked model leaked as a bit-level copy. Recent attacks on stealing models yet shown the possibility to leak a model by approximating it [33] through tailored queries. A crucial perspective is thus to investigate watermark techniques that can resist this alternative type of attacks.

Acknowledgements The authors would like to thank the reviewers for their constructive comments.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015). URL `https://www.tensorflow.org/`. Software available from tensorflow.org
2. Adi, Y., Baum, C., Cisse, M., Pinkas, B., Keshet, J.: Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In: 27th {USENIX} Security Symposium ({USENIX} Security 18), pp. 1615–1631 (2018)
3. van den Berg, E.: Some insights into the geometry and training of neural networks. arXiv preprint arXiv:1605.00329 (2016)
4. Braudaway, G.W., Magerlein, K.A., C.Mintzer, F.: Color correct digital watermarking of images. In: United States Patent 5530759 (1996)
5. Carlini, N., Wagner, D.A.: Audio adversarial examples: Targeted attacks on speech-to-text. CoRR **abs/1801.01944** (2018). URL `http://arxiv.org/abs/1801.01944`
6. Chang, C.Y., Su, S.J.: A neural-network-based robust watermarking scheme. In: SMC (2005)
7. Chollet, F., et al.: Keras. `https://keras.io` (2015)
8. Davchev, T., Korres, T., Fotiadis, S., Antonopoulos, N., Ramamoorthy, S.: An empirical evaluation of adversarial robustness under transfer learning. In: ICML Workshop on Understanding and Improving General-ization in Deep Learning (2019)
9. Duddu, V., Samanta, D., Rao, D.V., Balas, V.E.: Stealing neural networks via timing side channels. CoRR **abs/1812.11720** (2018). URL `http://arxiv.org/abs/1812.11720`
10. Duddu, V., Samanta, D., Rao, D.V., Balas, V.E.: Stealing neural networks via timing side channels. CoRR **abs/1812.11720** (2018). URL `http://arxiv.org/abs/1812.11720`
11. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: ICLR (2015)
12. Grosse, K., Manoharan, P., Papernot, N., Backes, M., McDaniel, P.D.: On the (statistical) detection of adversarial examples. CoRR **abs/1702.06280** (2017)
13. Guo, J., Potkonjak, M.: Watermarking deep neural networks for embedded systems. In: 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–8 (2018). DOI 10.1145/3240765.3240862
14. Hartung, F., Kutter, M.: Multimedia watermarking techniques. Proceedings of the IEEE **87**(7), 1079–1107 (1999). DOI 10.1109/5.771066
15. Le, Q.V., Jaitly, N., Hinton, G.E.: A simple way to initialize recurrent networks of rectified linear units. CoRR **abs/1504.00941** (2015). URL `http://arxiv.org/abs/1504.00941`
16. Le Merrer, E., Perez, P., Trédan, G.: Adversarial frontier stitching for remote neural network watermarking. CoRR **abs/1711.01894** (2017). URL `http://arxiv.org/abs/1711.01894`
17. Le Merrer, E., Trédan, G.: Tampernn: Efficient tampering detection of deployed neural nets. CoRR **abs/1903.00317** (2019)
18. LeCun, Y., Cortes, C., Burges, C.J.: The mnist database of handwritten digits. `http://yann.lecun.com/exdb/mnist` (1998)
19. Li, S., Neupane, A., Paul, S., Song, C., Krishnamurthy, S.V., Roy-Chowdhury, A.K., Swami, A.: Adversarial perturbations against real-time video classification systems. CoRR **abs/1807.00458** (2018). URL `http://arxiv.org/abs/1807.00458`
20. Liu, Y., Ma, S., Aafer, Y., Lee, W.C., Zhai, J., Wang, W., Zhang, X.: Trojaning attack on neural networks. In: NDSS (2017)
21. Moosavi-Dezfooli, S., Fawzi, A., Fawzi, O., Frossard, P.: Universal adversarial perturbations. In: CVPR (2017)
22. Nagai, Y., Uchida, Y., Sakazawa, S., Satoh, S.: Digital watermarking for deep neural networks. International Journal of Multimedia Information Retrieval **7**(1), 3–16 (2018)
23. Oh, S.J., Augustin, M., Fritz, M., Schiele, B.: Towards reverse-engineering black-box neural networks. In: International Conference on Learning Representations (2018). URL `https://openreview.net/forum?id=BydjJte0-`
24. Papernot, N., Carlini, N., Goodfellow, I., Feinman, R., Faghri, F., Matyasko, A., Hambardzumyan, K., Juang, Y.L., Kurakin, A., Sheatsley, R., Garg, A., Lin, Y.C.: cleverhans v2.0.0: an adversarial machine learning library. arXiv preprint arXiv:1610.00768 (2017)
25. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against machine learning. In: ASIA CCS (2017)
26. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Berkay Celik, Z., Swami, A.: The Limitations of Deep Learning in Adversarial Settings. arXiv preprint arXiv:1511.07528 (2015)
27. Papernot, N., McDaniel, P.D., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. arXiv preprint arXiv:1511.07528 (2015)
28. Rouhani, B.D., Chen, H., Koushanfar, F.: Deepsigns: A generic watermarking framework for IP protection of deep learning models. CoRR **abs/1804.00750** (2018). URL `http://arxiv.org/abs/1804.00750`
29. Rozsa, A., Günther, M., Boult, T.E.: Are accuracy and robustness correlated? In: ICMLA (2016)
30. Sethi, T.S., Kantardzic, M.: Data driven exploratory attacks on black box classifiers in adversarial domains. Neurocomputing **289**, 129 – 143 (2018). DOI https://doi.org/10.1016/j.neucom.2018.02.007. URL `http://www.sciencedirect.com/science/article/pii/S092523121830136X`

31. Shafahi, A., Huang, W.R., Studer, C., Feizi, S., Goldstein, T.: Are adversarial examples inevitable? CoRR **abs/1809.02104** (2018). URL http://arxiv.org/abs/1809.02104
32. Shin, H.C., Roth, H.R., Gao, M., Lu, L., Xu, Z., Nogues, I., Yao, J., Mollura, D., Summers, R.M.: Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. IEEE Transactions on Medical Imaging **35**(5), 1285–1298 (2016). DOI 10.1109/TMI.2016.2528162
33. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction apis. In: USENIX Security Symposium (2016)
34. Tramèr, F., Kurakin, A., Papernot, N., Boneh, D., McDaniel, P.: Ensemble adversarial training: Attacks and defenses. arXiv preprint arXiv:1705.07204 (2017)
35. Uchida, Y., Nagai, Y., Sakazawa, S., Satoh, S.: Embedding watermarks into deep neural networks. In: ICMR (2017)
36. Van Schyndel, R.G., Tirkel, A.Z., Osborne, C.F.: A digital watermark. In: Proceedings of 1st International Conference on Image Processing, vol. 2, pp. 86–90. IEEE (1994)
37. Wang, B., Gong, N.Z.: Stealing hyperparameters in machine learning. CoRR **abs/1802.05351** (2018). URL http://arxiv.org/abs/1802.05351
38. Yuan, X., He, P., Zhu, Q., Li, X.: Adversarial examples: Attacks and defenses for deep learning. IEEE Transactions on Neural Networks and Learning Systems pp. 1–20 (2019). DOI 10.1109/TNNLS.2018.2886017
39. Zhang, J., Gu, Z., Jang, J., Wu, H., Stoecklin, M.P., Huang, H., Molloy, I.: Protecting intellectual property of deep neural networks with watermarking. In: Proceedings of the 2018 on Asia Conference on Computer and Communications Security, pp. 159–172. ACM (2018)
40. Zhao, X., Liu, Q., Zheng, H., Zhao, B.Y.: Towards graph watermarks. In: COSN (2015)